# Marimba: A Tool for Verifying Properties of Hidden Markov Models

Noé Hernández[1], Kerstin Eder[3,4], Evgeni Magid[3,4], Jesús Savage[2], and David A. Rosenblueth[1]

[1] Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas
[2] Facultad de Ingeniería
Universidad Nacional Autónoma de México, D.F., México
no_hernan@ciencias.unam.mx,savage@servidor.unam.mx,drosenbl@unam.mx
[3] Department of Computer Science, University of Bristol, Bristol, BS8 1UB, UK
[4] Bristol Robotics Laboratory, Bristol, BS16 1QY, UK
{Kerstin.Eder,Evgeni.Magid}@bristol.ac.uk

**Abstract.** The formal verification of properties of Hidden Markov Models (HMMs) is highly desirable for gaining confidence in the correctness of the model and the corresponding system. A significant step towards HMM verification was the development by Zhang et al. of a family of logics for verifying HMMs, called POCTL*, and its model checking algorithm. As far as we know, the verification tool we present here is the first one based on Zhang et al.'s approach. As an example of its effective application, we verify properties of a handover task in the context of human-robot interaction. Our tool was implemented in Haskell, and the experimental evaluation was performed using the humanoid robot Bert2.

## 1 Introduction

A Hidden Markov Model (HMM) is an extension of a Discrete Time Markov Chain (DTMC) where the states of the model are hidden but the observations are visible. Typically, an HMM is studied with respect to the three basic problems examined by Rabiner in [1]. However, to the best of our knowledge, no actual model checker exists for HMMs despite their broad range of applications, e.g., speech recognition, DNA sequence analysis, text recognition and robot control. We describe in this paper a tool for verifying HMM properties written in the Probabilistic Observation Computational Tree Logic* (POCTL* [2]), and use this tool for verifying properties of a robot-to-human handover interaction.

POCTL* is a specification language for HMM properties. It is a probabilistic version of CTL* where a set of observations is attached to the *next* operator. Zhang et al. [2] sketched two model checking algorithms for POCTL*, an "automaton based" approach, and a "direct" approach. We opted for the direct approach both for its lower time complexity and for its clarity, facilitating its correctness proof and its implementation [3]. This approach produces a DTMC $\mathcal{D}$ and a Linear Temporal Logic (LTL) formula $\phi$. So the PRISM [4] model checker could be used to verify $\phi$ on $\mathcal{D}$. Such a model checker follows an approach whose

complexity is doubly exponential in $|\phi|$ and polynomial in $|\mathcal{D}|$, whereas the direct approach verifies $\phi$ on $\mathcal{D}$ with the method by Courcoubetis et al. [5] whose complexity is singly exponential in $|\phi|$ and polynomial in $|\mathcal{D}|$, which is also the final complexity of our tool. This latter method repeatedly constructs a DTMC and rewrites an LTL formula, such that one temporal operator is removed each time while preserving the probability of satisfaction.

We have named our model checker Marimba. A marimba is a xylophone-like musical instrument that is popular in south-east Mexico and Central America. Marimba was implemented in Haskell and compiled with GHCi. Our tool is available for download from `https://github.com/nohernan/Marimba`.

## 2 Tool Architecture and Implementation

Haskell was chosen to code this first version of Marimba since it allows us to work in a high-level abstract layer, by providing useful mechanisms like lazy evaluation and a pure functional paradigm. Furthermore, Haskell excels at managing recursion; this is a valuable aspect because recursive calls are made continuously throughout the execution.

Marimba features a command-line interface. Moreover, instead of working with a command window, a more user friendly and preferable execution is accomplished through the *Emacs* text editor extended with the `Haskell-mode`.

### 2.1 Marimba's Input and Modules

The first input is a `.hmm` file with the six elements of an HMM $\mathcal{H}$, namely a finite set of states $S$, a state transition probability matrix $A$, a finite set of observations $\Theta$, an observation probability matrix $B$, a function $L$ that maps states to sets of atomic propositions from a set $AP_{\mathcal{H}}$, and an initial probability distribution $\pi$ over $S$. The second input is a POCTL* state formula $\Phi$ typed in the command window according to the syntactic rules:

$$\Phi ::= \mathsf{true} \mid \mathsf{false} \mid a \mid (\neg\Phi) \mid (\Phi \vee \Phi) \mid (\Phi \wedge \Phi) \mid (\mathcal{P}_{\bowtie p}(\phi)),$$
$$\phi ::= \Phi \mid (\neg\phi) \mid (\phi \vee \phi) \mid (\phi \wedge \phi) \mid (\mathbf{X}_{\mathsf{o}}\phi) \mid (\phi \mathcal{U}^{\leq n}\phi) \mid (\phi \mathcal{U} \phi),$$

where $a \in AP_{\mathcal{H}}$, $\mathsf{o} \in \Theta$, $n \in \mathbb{N}$, $p \in [0,1]$, and $\bowtie \in \{\leq, <, \geq, >\}$. In addition, we define $\mathbf{X}_{\Omega}\phi$ as a shorthand for $\bigvee_{\mathsf{o}\in\Omega} \mathbf{X}_{\mathsf{o}}\phi$ provided $\Omega \subseteq \Theta$. We examine below the six Haskell modules that constitute Marimba.

*ModelChecker.hs* performs the initial computations of the model checker for POCTL*. It recursively finds a most nested state subformula of $\Phi$, not being a propositional variable, and the states of $\mathcal{H}$ that satisfy it. Finding the states that satisfy a state subformula is straightforward when such a subformula is propositional. If, however, the state subformula is probabilistic, the module *DirectApproach.hs* obtains the states satisfying this subformula. Next, we extend the labels of such states with a new atomic proposition $a$. In $\Phi$, the state subformula being addressed is replaced by $a$. The base case occurs when we reach a propositional variable, so we return the states that have it in their label.

*DirectApproach.hs* transforms the HMM $\mathcal{H}$ into a DTMC $\mathcal{D}$, and removes from the specification the observation set attached to the *next* operator $\mathbf{X}$ by generating a conjunction of the observation-free $\mathbf{X}$ with a new propositional variable. Thus, we obtain an LTL formula that is passed, together with $\mathcal{D}$, to the module *Courcoubetis.hs*. The new propositional variables are drawn from the power set of observations. Remarkably, it is not necessary to compute such a power set since the label of a state in $\mathcal{D}$ is easily calculated.

*Courcoubetis.hs* implements a modified version of the method by Courcoubetis et al. to find the probability that an LTL formula is satisfied in a DTMC. In this module, when dealing with the $\mathcal{U}$ and $\mathcal{U}^{\leq n}$ operators, we apply ideas from [6] for computing a partition of states of $\mathcal{D}$. Moreover, to handle the $\mathcal{U}$ operator we have to solve a linear equation system. To that end, we use the *linearEqSolver* library [7], which in turn executes the *Z3* theorem prover [8].

*Lexer.hs* and *Parser.hs* are in charge of the syntactic analysis of the input. Finally, *Main.hs* is loaded to start Marimba. This module manages the interaction with the user, and starts the computation by passing control to *ModelChecker.hs*.

In a typical execution, Marimba prompts the user to enter a `.hmm` file path. Next, our tool asks whether or not the user wants to take into account the initial distribution in the computation of the probability of satisfaction. This choice corresponds to opposite ideas presented in [5] and [2], i.e., the method by Courcoubetis et al. uses the initial distribution to define their probability measure, contrary to that defined by Zhang et al. Afterwards, a POCTL* formula has to be entered. Marimba returns the list of states satisfying this formula, and asks the user whether there are more formulas to be verified on the same model.

The `.hmm` file is simply a text file where the elements of an HMM are defined, e.g., the set of states is defined by the reserved word `States`, and if the model consists of five states, we write `States=5`. Likewise, POCTL* formulas have a natural writing, for example, $\mathcal{P}_{<0.1}(\mathbf{X}_{\{o_1\}}a)$ is typed as `P[<0.1](X_{1}a)`.

Our implementation of Marimba in Haskell makes extensive use of ordinary arrays, which are known for a lack of efficiency in this programming language [9]. Thus, Marimba presents limitations in practice when considering large models. To better deal with this situation, a future work would consist in coding Marimba in a language like Java and make it a symbolic model checker. Nevertheless, with a high degree of confidence, we can say that this current implementation correctly performs the steps dictated by our version of Zhang et al.'s algorithm, mostly because Haskell provides an abstract and formal coding framework.

## 3 Verification of a Human-Robot Interaction

We applied Marimba to a real-world example, namely the verification of the robot-to-human handover task [10] using the robot Bert2 [11] at the Bristol Robotics Laboratory (BRL). The robot's decision to release the object during the handover task is determined by an HMM [10]. Figure 1 presents the state diagram of the HMM corresponding to the basic handover interaction, where the label $L(s)$ is defined for each state $s$.

We initialise $A$, $B$ and $\pi$ of the HMM for later training as follows. Since the first state of the handover process is Robot not hold, the initial distribution $\pi$ favours this state above the others. The initialisation of matrix $A$ must encourage the transitions shown in Figure 1. To initialise $B$, we consider as observations the ordered pairs whose first and second components are the index and middle finger metacarpophalangeal joint motor current values, respectively. By the Cartesian product of these values, we obtain 56,404 observations. Since these observations are merged with the states to generate the DTMC passed to *Courcoubetis.hs*, and the size of a formula could grow considerably by associating the *next* operator with up to 56,404 observations, Marimba's execution is not practical under these circumstances. Vector quantisation [12] was used to reduce the number of observations to just 13. This method eliminates redundancy by grouping similar pairs in regions. So, we obtained 13 regions of the plane, which were regarded as the observations of the HMM and taken to initialise matrix $B$. Applying vector quantisation causes a loss in accuracy, as indicated by the computation of the root mean square error. However, we can effectively find observations that are likely to characterise each of the four states of this HMM (see first liveness property below, for example).

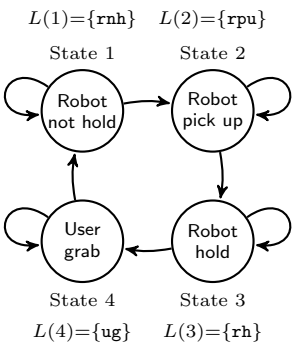


**Fig. 1.** The labelled states involved in the basic handover process.

To make reliable estimates, we collected observations from 50 handover experiments on Bert2. These observations were used to train the initial HMM with the reestimation method found in the solution of Rabiner's Problem 3 [1].

**Liveness properties.** A liveness property requires that a *good thing* happens during the execution of a system. For example, we would like to know whether *the model generates the sequence of observations* $\mathcal{O} = o_1, o_2, o_3, o_4$ *where* $o_1, o_2 \in \{3, 4, 6\}$ *and* $o_3, o_4 \in \{3, 4, 11\}$*, with probability greater than 0.88*, that is, $\mathcal{P}_{>0.88}(\mathbf{X}_{\{3,4,6\}}(\mathbf{X}_{\{3,4,6\}}(\mathbf{X}_{\{3,4,11\}}(\mathbf{X}_{\{3,4,11\}}\mathsf{true}))))$. Interestingly, this property is a generalisation of Rabiner's Problem 1 [1]. Marimba's execution for this property is found in Figure 2. The inputs are the trained HMM, defined in `ModelBert2.hmm`, and the previous formula. The output returned by Marimba is State 4. Hence, the model starting at state User grab is likely to generate $\mathcal{O}$.

A second liveness property states that *with probability at least 0.9,* Bert2 *releases the object when the user grabs it.* The POCTL* formula for this property is $\mathcal{P}_{\geq 0.9}(\mathtt{rh} \wedge (\mathtt{rh}\ \mathcal{U}\ (\mathtt{ug} \wedge \mathtt{ug}\ \mathcal{U}\ \mathtt{rnh})))$. Marimba outputs State 3, i.e., the specification is satisfied when the starting state is Robot hold. So, we expect Bert2 to hold the object, and let it go when the user grabs it.

**Safety properties.** A safety property establishes that a *bad thing* does not occur during the execution of a system. For instance, *with probability less than 0.05,* Bert2 *abandons its serving position with the user not grabbing the object*, that is, $\mathcal{P}_{<0.05}(\mathtt{rh} \wedge \mathbf{X}_{\Theta}(\mathtt{rnh} \vee \mathtt{rpu}))$, where $\Theta$ is the set of observations. Our

```
Main> main
Enter the file name where the HMM is located.
examples/ModelBert2.hmm
Would you like to consider each state as if it were the initial
state, i.e., as if it had initial distribution value equal to 1? y/n: y
Enter the POCTL* formula we are interested in.
P[>0.88] (X_{3,4,6}(X_{3,4,6}(X_{3,4,11}(X_{3,4,11}T))))
The states that satisfy it are:
(Probability of satisfaction of each state:[4.998198505964186e-10,
            4.08659792160621e-6,7.508994137303159e-3,0.8915357419467848])
[4]
Do you want to continue checking more specifications? y/n: n
```

**Fig. 2.** Verifying a property with Marimba.

model checker returns $\{1, 2, 3, 4\}$ as the set of states satisfying this property. We conclude that it is unlikely that the model, being at state Robot hold, reaches a state other than User grab, that is, Robot not hold or Robot pick up.

The satisfaction of the previous three specifications provides us with confidence that Bert2 reliably performs the handover interaction specified above.

On an Intel® Core™ i3 1.70GHz computer with 4GB in memory, Marimba takes 28.55s to compute the states satisfying the first liveness formula. The time required for checking the other two properties studied here is around 0.06s.

Further examples are given in the *examples* folder and *user's manual* that come with Marimba's source code.

## 4 Conclusions

Since the automatic verification of properties of HMMs seems to be an unattended problem, we present here Marimba, a Haskell implementation of the model checking algorithm for POCTL* [2]. This model checking algorithm was slightly modified to carry out its computations in a real program. Marimba's calculation is basically broken out in three stages that are coded in the modules *ModelChecker.hs*, *DirectApproach.hs* and *Courcoubetis.hs*, such that the involved components, steps and transformations are well arranged throughout the implementation. Finally, we have successfully applied Marimba to verify relevant properties of a handover interaction from the robot Bert2 to a human.

**Acknowledgements.** We gratefully acknowledge support from grants PAPIIT IN113013 and Conacyt 221341, and especially thank the BRL staff for their assistance operating the robot Bert2. E. Magid and K. Eder have been supported, in full and in part, respectively, by the UK EPSRC grant EP/K006320/1 ROBOSAFE: "Trustworthy Robotic Assistants".

## References

1. Rabiner, L.R.: A tutorial on hidden Markov models and selected applications in speech recognition. Proceedings of the IEEE **77** (1989) 257–286

2. Zhang, L., Hermanns, H., Jansen, D.N.: Logic and model checking for hidden Markov models. In: Formal Techniques for Networked and Distributed Systems, FORTE 2005. Volume 3731 of LNCS., Springer (2005) 98–112
3. Hernández, N.: Model checking based on the hidden Markov model and its application to human-robot interaction. Master's thesis, Universidad Nacional Autónoma de México, México (2014) Available from `http://132.248.9.195/ptd2014/noviembre/303087692/Index.html`.
4. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Proc. 23rd International Conference on Computer Aided Verification (CAV '11). Volume 6806 of LNCS., Springer (2011) 585–591
5. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. J. ACM **42**(4) (July 1995) 857–907
6. Rutten, J., Kwiatkowska, M., Norman, G., Parker, D.: Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems. Volume 23 of CRM Monograph Series. American Mathematical Society (2004)
7. Erkok, L.: linearEqSolver: a library to solve systems of linear equations, using SMT solvers. `https://github.com/LeventErkok/linearEqSolver`
8. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Proceedings of the Theory and Practice of Software (TACAS '08). LNCS, Springer (2008) 337–340
9. Chakravarty, M.M.T., Keller, G.: An Approach to Fast Arrays in Haskell. In: Lecture notes for The Summer School and Workshop on Advanced Functional Programming 2002. Volume 2638 of LNCS. (2003) 27–58
10. Grigore, E.C., Eder, K., Pipe, A.G., Melhuish, C., Leonards, U.: Joint action understanding improves robot-to-human object handover. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE (2013) 4622–4629
11. Lenz, A., Skachek, S., Hamann, K., Steinwender, J., Pipe, A.G., Melhuish, C.: The BERT2 infrastructure: An integrated system for the study of human-robot interaction. In: 10th IEEE-RAS International Conference on Humanoid Robots, IEEE (2010) 346–351
12. Linde, Y., Buzo, A., Gray, R.M.: An algorithm for vector quantizer design. IEEE Transactions on Communications **28** (1980) 84–95